May 02, 00 16:13 **SearchTree.i3** Page 1/1

```
    INTERFACE SearchTree ;

    TYPE Item = INTEGER ;
         Node = REF RECORD
             key         : Item ;
             left, right : Node ;
         END ;

    VAR root : Node ;

    PROCEDURE Init() ;
    PROCEDURE Insert(key : Item) ;
    PROCEDURE Search(key : Item) : BOOLEAN ;
    PROCEDURE Write() ;

    END SearchTree .
```

May 02, 00 16:13 **SearchTree.m3** Page

```
    MODULE SearchTree ;

    IMPORT IO ;


    PROCEDURE Init() =
    BEGIN
        root := NIL ;
    END Init ;


    PROCEDURE Insert(key : Item) =
    VAR p, n : Node ;
    BEGIN
        n := NEW(Node) ;
        n^.key   := key ;
        n^.left  := NIL ;
        n^.right := NIL ;

        IF root = NIL THEN
            root := n ;
        ELSE
            p := root ;
            LOOP
                IF key <= p^.key THEN
                    IF p^.left = NIL THEN
                        p^.left := n ;
                        RETURN ;
                    ELSE
                        p := p^.left ;
                    END ;
                ELSE (* key > p^.key *)
                    IF p^.right = NIL THEN
                        p^.right := n ;
                        RETURN ;
                    ELSE
                        p := p^.right ;
                    END ;
                END ;
            END ;
        END ;
    END Insert ;


    PROCEDURE Search(key : Item) : BOOLEAN =
    VAR n : Node ;
    BEGIN
        n := root ;
        WHILE n # NIL DO
            IF key = n^.key THEN
                RETURN TRUE ;
            ELSIF key <= n^.key THEN
                n := n^.left ;
            ELSE (* key > n^.key *)
                n := n^.right ;
            END ;
        END ;
        RETURN FALSE ;
    END Search ;


    PROCEDURE WriteSubTree(n : Node) =
    BEGIN
        IF n # NIL THEN
            WriteSubTree(n^.left) ;
```

```
            IO.PutInt(n^.key) ;
            IO.Put("\n") ;

            WriteSubTree(n^.right) ;
        END;
    END WriteSubTree ;


    PROCEDURE Write() =
    BEGIN
        WriteSubTree(root) ;
    END Write ;



    BEGIN
    END SearchTree .
```



```
    MODULE Main ;

    IMPORT IO, SearchTree, Traversal ;


    VAR i : INTEGER ;
    BEGIN
        SearchTree.Init() ;

        WHILE NOT IO.EOF() DO
            TRY
                i := IO.GetInt() ;
            EXCEPT
              IO.Error => EXIT ;
            END ;
            SearchTree.Insert(i) ;
        END ;

        IO.Put("Rekursive InOrder-Traversierung:\n") ;
        SearchTree.Write() ;

        IO.Put("\n\nlineares Durchsuchen:\n") ;
        FOR i := -1000 TO 1000 DO
            IF SearchTree.Search(i) THEN
                IO.PutInt(i) ;
                IO.Put("\n") ;
            END ;
        END ;

        IO.Put("\n\nPreOrder-Traversierung:\n") ;
        Traversal.TraversePreOrder() ;
        IO.Put("\n\nInOrder-Traversierung:\n") ;
        Traversal.TraverseInOrder() ;
        IO.Put("\n\nPostOrder-Traversierung:\n") ;
        Traversal.TraversePostOrder() ;
        IO.Put("\n\nLevelOrder-Traversierung:\n") ;
        Traversal.TraverseLevelOrder() ;
    END Main .
```

May 02, 00 17:22 **Stack.i3** Page 1/1

```
INTERFACE Stack ;

IMPORT SearchTree ;


CONST Capacity = 100 ;

TYPE Item = RECORD
    node : SearchTree.Node ;
    flag : BOOLEAN ;
END ;

PROCEDURE Push(item : Item) ;
PROCEDURE Pop() : Item ;
PROCEDURE IsFull() : BOOLEAN ;
PROCEDURE IsEmpty() : BOOLEAN ;

END Stack .
```

May 02, 00 16:20 **Stack.m3** Page

```
MODULE Stack ;

TYPE Index = [0..Capacity] ;

VAR store : ARRAY Index OF Item ;
    top   : Index ;


PROCEDURE Push(item : Item) =
BEGIN
    <* ASSERT NOT IsFull() *>
    store[top] := item ;
    INC(top) ;
END Push ;


PROCEDURE Pop() : Item =
BEGIN
    <* ASSERT NOT IsEmpty() *>
    DEC(top) ;
    RETURN store[top] ;
END Pop ;


PROCEDURE IsFull() : BOOLEAN =
BEGIN
    RETURN top > Capacity ;
END IsFull ;


PROCEDURE IsEmpty() : BOOLEAN =
BEGIN
    RETURN top <= 0 ;
END IsEmpty ;


BEGIN
    top := 0 ;
END Stack .
```

May 02, 00 15:47 **Queue.i3** Page 1/1

```
INTERFACE Queue ;

IMPORT SearchTree ;


CONST Capacity = 100 ;

TYPE Item = SearchTree.Node ;

PROCEDURE Put(item : Item) ;
PROCEDURE Get() : Item ;
PROCEDURE IsFull() : BOOLEAN ;
PROCEDURE IsEmpty() : BOOLEAN ;

END Queue .
```

Apr 28, 00 17:03 **Queue.m3** Page

```
MODULE Queue ;

TYPE Index = [0..Capacity] ;

VAR store      : ARRAY Index OF Item ;
    head, tail : Index ;


PROCEDURE Put(item : Item) =
BEGIN
    <* ASSERT NOT IsFull() *>

    store[tail] := item ;

    IF (tail < Capacity) THEN
        INC(tail) ;
    ELSE
        tail := 0 ;
    END ;
END Put ;


PROCEDURE Get() : Item =
VAR item : Item ;
BEGIN
    <* ASSERT NOT IsEmpty() *>

    item := store[head] ;

    IF (head < Capacity) THEN
        INC(head) ;
    ELSE
        head := 0 ;
    END ;

    RETURN item ;
END Get ;


PROCEDURE IsFull() : BOOLEAN =
BEGIN
    IF head = 0 THEN
        RETURN tail = Capacity ;
    ELSE
        RETURN tail = head - 1 ;
    END ;
END IsFull ;


PROCEDURE IsEmpty() : BOOLEAN =
BEGIN
    RETURN head = tail ;
END IsEmpty ;


BEGIN
    head := 0 ;
    tail := 0 ;
END Queue .
```

Apr 28, 00 15:27 **Traversal.i3** Page 1/1

```
    INTERFACE Traversal ;

    PROCEDURE TraversePreOrder() ;
    PROCEDURE TraversePostOrder() ;
    PROCEDURE TraverseInOrder() ;
    PROCEDURE TraverseLevelOrder() ;

    END Traversal .
```

May 02, 00 17:22 **Traversal.m3** Page

```
    MODULE Traversal ;

    IMPORT IO, Stack, Queue ;
    FROM SearchTree IMPORT Node, root ;


    PROCEDURE Visit(n : Node) =
    BEGIN
        IO.PutInt(n^.key) ;
        IO.Put("\n") ;
    END Visit ;


    PROCEDURE TraversePreOrder() =
    VAR n : Node ;
    BEGIN
        Stack.Push(Stack.Item{root, TRUE}) ;
        WHILE NOT Stack.IsEmpty() DO
            n := Stack.Pop().node ;
            IF n # NIL THEN
                Visit(n) ;
                Stack.Push(Stack.Item{n^.right, TRUE}) ;
                Stack.Push(Stack.Item{n^.left,  TRUE}) ;
            END ;
        END ;
    END TraversePreOrder ;


    PROCEDURE TraverseInOrder() =
    VAR s : Stack.Item ;
    BEGIN
        Stack.Push(Stack.Item{root, TRUE}) ;
        WHILE NOT Stack.IsEmpty() DO
            s := Stack.Pop() ;
            IF s.node # NIL THEN
                IF s.flag THEN
                    Stack.Push(Stack.Item{s.node, FALSE}) ;
                    Stack.Push(Stack.Item{s.node^.left, TRUE}) ;
                ELSE
                    Visit(s.node) ;
                    Stack.Push(Stack.Item{s.node^.right, TRUE}) ;
                END ;
            END ;
        END ;
    END TraverseInOrder ;


    PROCEDURE TraversePostOrder() =
    VAR s : Stack.Item ;
    BEGIN
        Stack.Push(Stack.Item{root, TRUE}) ;
        WHILE NOT Stack.IsEmpty() DO
            s := Stack.Pop() ;
            IF s.node # NIL THEN
                IF s.flag THEN
                    Stack.Push(Stack.Item{s.node, FALSE}) ;
                    Stack.Push(Stack.Item{s.node^.right, TRUE}) ;
                    Stack.Push(Stack.Item{s.node^.left, TRUE}) ;
                ELSE (* s.flag = 1 *)
                    Visit(s.node) ;
                END ;
            END ;
        END ;
    END TraversePostOrder ;

```

```
    PROCEDURE TraverseLevelOrder() =
    VAR n : Node ;
    BEGIN
70      Queue.Put(root) ;
        WHILE NOT Queue.IsEmpty() DO
            n := Queue.Get() ;
            IF n # NIL THEN
                Visit(n) ;
75              Queue.Put(n^.left) ;
                Queue.Put(n^.right) ;
            END ;
        END ;
    END TraverseLevelOrder ;
80

    BEGIN
    END Traversal .
```



```
    Rekursive InOrder-Traversierung:
    -798
    -611
    -495
5   -299
    -222
    -125
    -105
    -100
10  -89
    -34
    -30
    -30
    -8
15  -5
    -3
    2
    10
    13
20  18
    20
    100
    105
    121
25  200
    300
    450
    500
    501
30

    lineares Durchsuchen:
    -798
    -611
35  -495
    -299
    -222
    -125
    -105
40  -100
    -89
    -34
    -30
    -8
45  -5
    -3
    2
    10
    13
50  18
    20
    100
    105
    121
55  200
    300
    450
    500
    501
60

    PreOrder-Traversierung:
    18
    -5
65  -8
    -30
```

```
-89
-100
-105
-611
-798
-222
-299
-495
-125
-34
-30
-3
10
2
13
121
100
20
105
200
300
500
450
501


InOrder-Traversierung:
-798
-611
-495
-299
-222
-125
-105
-100
-89
-34
-30
-30
-8
-5
-3
2
10
13
18
20
100
105
121
200
300
450
500
501


PostOrder-Traversierung:
-798
-495
-299
-125
-222
-611
-105
-100
```



```
-30
-34
-89
-30
-8
2
13
10
-3
-5
20
105
100
450
501
500
300
200
121
18


LevelOrder-Traversierung:
18
-5
121
-8
-3
100
200
-30
10
20
105
300
-89
2
13
500
-100
-34
450
501
-105
-30
-611
-798
-222
-299
-125
-495
```